```c
//
// 7_1.cpp
//

#include <stdio.h>
#include <stdlib.h>
#include <math.h>

const int N = 4;                // 行列の行数
const double EPS = 1e-6;        // ピボットが 0 かどうか判定する値
typedef double Matrix[N][N+1];  // 行列の型を定義

// 行列を表示する関数
void print_mat(Matrix A){
    for(int i=0; i<N; i++){
        for(int j=0; j<N; j++){
            printf("%8.3f", A[i][j]);
        }
        printf("  |%8.3f\n", A[i][N]);
    }
    printf("\n");
}

// csv ファイルから A_mat と b_vec を読み込み、拡大行列を作る関数
void read_A_b(char* A_file, char* b_file, Matrix A1, Matrix A2){
    FILE* A_fp = fopen(A_file, "rt");   // A_mat の入力ファイルポインタ
    FILE* b_fp = fopen(b_file, "rt");   // b_vec の入力ファイルポインタ

    if(!A_fp){  // A_mat が開けなかったらエラー
        fprintf(stderr, "Can't open file %s\n", A_file);
        exit(2);
    }

    if(!b_fp){  // b_vec が開けなかったらエラー
        fprintf(stderr, "Can't open file %s\n", b_file);
        exit(3);
    }

    for(int i=0; i<N; i++){
        for(int j=0; j<N; j++){
            fscanf(A_fp, "%lf,", &A1[i][j]);    // A_mat は 1~N 列目
            A2[i][j] = A1[i][j];
        }
        fscanf(b_fp, "%lf,", &A1[i][N]);    // b_vec は N+1 列目
        A2[i][N] = A1[i][N];
    }
    print_mat(A1);  // 行列を表示

    fclose(A_fp);   // 入力ファイル(A_mat)を閉じる
    fclose(b_fp);   // 入力ファイル(b_vec)を閉じる
}

// 最大値を検索し、その行数を返す関数
int search_max(Matrix A, int k){
    double max = 0.0;   // 最大値を格納する変数
    int max_row = k;    // 最大値がある行数を格納する変数
    for(int i=k; i<N; i++){
        double val = fabs(A[i][k]);
        if(val>EPS && val>fabs(max)){
            max = A[i][k];
            max_row = i;
        }
    }
    return max_row; // 最大値がある行数を返す
}

// 行を入れ替える関数
void change_row(Matrix A, int i, int max_row){
    for(int j=0; j<N+1; j++){
        double tmp = A[i][j];   // 一時的に格納する変数
        A[i][j] = A[max_row][j];
        A[max_row][j] = tmp;
    }
    printf("Changed row %d and row %d.\n", i+1, max_row+1);
    print_mat(A);   // 行列を表示
}
```

```
77
78  // ガウス・ジョルダン法
79  int gauss_jordan(Matrix A){
80      int rankcount = 0;
81      for(int i=0; i<N; i++){
82          int max_row = search_max(A, i); // 最大値がある行数を検索
83          if(fabs(A[i][i]) < EPS){
84              for(int j=i+1; j<N; j++){
85                  if(fabs(A[j][i]) > EPS){
86                      change_row(A, i, j);
87                  break;
88                  }
89              }
90          }
91          double pivot = A[i][i]; // ピボットを定義
92          if(fabs(pivot) > EPS){ // ピボットが 0 でなければ
93              for(int j=i; j<N+1; j++){   // 各列をピボットで除算
94                  A[i][j] /= pivot;
95              }
96              for(int k=0; k<N; k++){ // 掃き出し計算
97                  if(k == i){
98                      continue;
99                  }
100                 double tmp = A[k][i];   // 一時的に値を保存する変数
101                 for(int j=i; j<N+1; j++){
102                     A[k][j] -= tmp * A[i][j];
103                 }
104             }
105             print_mat(A);   // 行列を表示
106         }else{
107             rankcount++;    // ピボットが 0 であれば1つランク落ち
108         }
109     }
110     return N - rankcount;
111 }
112
113 // 前進消去
114 int forward_elimination(Matrix A){
115     int rankcount = 0;
116     for(int i=0; i<N; i++){
117         int max_row = search_max(A, i); // 最大値がある行数を検索
118         if(i != max_row){
119             change_row(A, i, max_row);  // 最大値がある行数と行を入れ替え
120         }
121         double pivot = A[i][i];
122         if(fabs(pivot) > EPS){ // ピボットが 0 でなければ
123             for(int j=i; j<N+1; j++){ // 各列をピボットで除算
124                 A[i][j] /= pivot;
125             }
126             for(int k=i+1; k<N; k++){   // 掃き出し計算
127                 double tmp = A[k][i];   // 一時的に値を保存する変数
128                 for(int j=i; j<N+1; j++){
129                     A[k][j] -= tmp * A[i][j];
130                 }
131             }
132             print_mat(A);   // 行列を表示
133         }else{
134             rankcount++;    // ピボットが 0 であれば1つランク落ち
135         }
136     }
137     return N - rankcount;
138 }
139
140 // 後退代入
141 void backward_substitution(Matrix A, int rank){
142     if(rank < N){   // 行数と階数が一致していれば後退代入
143         return;
144     }
145     printf("x[%d] = %6.3f\n", N, A[N-1][N]);    // 最後の行の解は計算済み
146     for(int i=N-2; i>=0; i--){
147         for(int j=N-1; j>i; j--){
148             A[i][N] -= A[i][j]*A[j][N];;
149         }
150         printf("x[%d] = %6.3f\n", i+1, A[i][N]);    // 解を表示
151     }
152 }
153
```

```
// ガウス消去法
void gauss_elimination(Matrix A, int& rank){
    printf(" < Forward elimination >\n");
    rank = forward_elimination(A);  // 前進消去
    printf(" < Backward substitution >\n");
    backward_substitution(A, rank); // 後退代入
}

// 解を表示する関数
void print_ans(Matrix A, int rank){
    if(rank == N){  // 行数と階数が一致していれば解を表示
        printf("\n----- Answers -----\n");
        for(int i=0; i<N; i++){
            printf("x[%d] = %6.3f\n", i+1, A[i][N]);
        }
    }else{
        printf("This A_mat is singular.\t(Rank = %d)\n", rank); // 行数と階数が一致しなければ階数を表示
    }
    printf("\n");
}

// メイン関数
int main(int argc, char* argv[]){
    if(argc!=3){            //実数時引数が２個なかったらエラーとする
        fprintf(stderr, "Usage: %s A_mat_file b_vec_file\n", argv[0]);
        exit(1);
    }
    char* A_file = argv[1]; //パラメータの１番目を入力ファイル名
    char* b_file = argv[2]; //パラメータの２番目を入力ファイル名
    Matrix A1 = {}; // 拡大行列を定義
    Matrix A2 = {}; // 拡大行列を定義
    read_A_b(A_file, b_file, A1, A2);   // csv ファイルから A_mat と b_vec を読み込み、拡大行列を作成

    int rank1 = 0;  // 階数を定義
    int rank2 = 0;  // 階数を定義

    printf("\n----- Solution by the Gauss Jordan method -----\n");
    rank1 =  gauss_jordan(A1);
    print_ans(A1, rank1);   // 連立方程式の解を表示

    printf("\n----- Solution by the Gauss Elimination method -----\n");
    gauss_elimination(A2, rank2);   // ガウス消去法で連立方程式の解を計算
    print_ans(A2, rank2);   // 連立方程式の解を表示

    return 0;
}
```